Parallel GP interface tutorial

B. Allombert

IMB
CNRS/Université Bordeaux 1

07/01/2014

Introduction

PARI now supports two common multi-threading technologies :

- POSIX thread : run on a single machine, lightweight, flexible, fragile.
- Message passing interface (MPI) : run on as many machine as you want, robust, rigid, heavyweight. Used by most clusters.

However the parallel GP interface does not depend on the multithread interface : a properly written GP program will work identically with both. In this tutorial we will focus on POSIX threads.

POSIX threads

- To use POSIX threads, add `-mt=pthread`
- It can be useful to add `-time=ftime` so that the GP timer report real time instead of cumulated CPU time.
- `gp-sta` is generally 20% faster than gp-dyn.

```
./Configure --mt=pthread --time=ftime
make test-parallel
make test-rnfkummer
```

Compare `gp-sta` against `gp-dyn`.

```
ln -s Olinux-x86_64/gp-sta .
./gp-sta
```

Check for "threading engine : pthread"

Resources

The number of secondary threads to use is controlled by `default(nbthreads)`. The default value of nbthreads is the number of CPU threads (i.e. the number of CPU cores multiplied by the hyperthreading factor). The default can be freely modified.
The PARI stack size in secondary threads is controlled by `default(threadsize)`, so the total memory allocated is equal to $\texttt{parisize} + \texttt{nbthreads} \times \texttt{threadsize}$. By default, $\texttt{threadsize} = \texttt{parisize}$.

`default(nbthreads)`

Concept

GP provides functions that allows parallel execution of GP code, subject to the following limitations : the parallel code

- must not access global variables or local variables declared with `local()` (but `my()` is OK),
- must be free of side effect.

Simple examples

```
ismersenne(x)=ispseudoprime(2^x-1);
default(timer,1);
apply(ismersenne,primes(400))
parapply(ismersenne,primes(400))
select(ismersenne,primes(400))
parselect(ismersenne,primes(400))
```

Avoiding global variables

```
V=primes(400);
parvector(#V,i,ispseudoprime(2^V[i]-1))
  *** parvector: mt: global variable not
  *** supported: V.
break
fun(V)=parvector(#V,i,ispseudoprime(2^V[i]-1));
fun(V)
my(V=V);parvector(#V,i,ispseudoprime(2^V[i]-1))
```

Avoiding global functions

```
ismersenne(x)=ispseudoprime(2^x-1);
fun(V)=parvector(#V,i,ismersenne(V[i]));
fun(primes(400))
  *** parvector: mt: global variable not
  *** supported: ismersenne.
break
fun(V)=
{ my(ismersenne=ismersenne);
  parvector(#V,i,ismersenne(V[i]));
}
fun(primes(400))
```

Using inline

```
inline(ismersenne);
ismersenne(x)=ispseudoprime(2^x-1);
fun(V)=parvector(#V,i,ismersenne(V[i]));
fun(primes(400))
```

using polynomial indeterminates

```
fun(n)=bnfinit(x^n-2).no;
parapply(fun,[1..30])
  *** parapply: mt: global variable not
  *** supported: x.
break
fun(n)=bnfinit('x^n-2).no;
default(timer,1);
default(parisize,"16M");
apply(fun,[1..30])
parapply(fun,[1..30])
parapply(fun,-[-30..-1])
```

Grouping small tasks

```
inline(thuemorse);
thuemorse(n)=
 my(V=binary(n)); (-1)^sum(i=1,#V,V[i]);
default(timer,1);
sum(n=1,2*10^6, thuemorse(n)/n*1.)
parsum(n=1,2*10^6, thuemorse(n)/n*1.)
parsum(N=1,200, \
  sum(n=1+(N-1)*10^4, N*10^4, thuemorse(n)/n*1.))
```

Using parfor/parforprime

```
inline(ismersenne);
ismersenne(x)=ispseudoprime(2^x-1);
parforprime(p=1,999,ismersenne(p),c,if(c,print(p)))
prodmersenne(N)=
{ my(R=1);
  parforprime(p=1,N,
    ismersenne(p),
    c,
    if(c, R*=p));
  R;
}
prodmersenne(1000)
```

```
inline(ismersenne);
ismersenne(x)=ispseudoprime(2^x-1);
findmersenne(a)=
  parforprime(p=a,,ismersenne(p),c,if(c,return(p)))
findmersenne(4000)
findmersenne(8)
findmersenne(8)
```

```
inline(ismersenne);
ismersenne(x)=ispseudoprime(2^x-1);
parfirst(fun,V)=
  parfor(i=1,#V,fun(V[i]),j,if(j,return([i,j])));
parfirst(ismersenne,[4001..5000])
```